Graph Visualization Language (GVL) Proposal

Minhe Zhang (mz2864)
Yaxin Chen (yc3995)
Aster Wei (aw3389)
Jiawen Yan (jy3088)

October 2021

1 Introduction

Our team is planning to implement a imperative, C-like, and strong typing language which is specialized to visualize graph data structures and algorithms. It should support primitive data type like *int*, *float*, *boolean*, and *char*. It should also support complex data type like *array* and *struct*. Most importantly, user should be able to define and operate essential components of graph including *node* and *edge*.
As our language is specialized to represent graph, programmer should conveniently define graphs and manipulate their components such as nodes and edges so that graph algorithms can be implemented and represented easily.
For visualization, we have decided to use OpenGL as our visualization library so far. Some basic graphic functionalities are drawing the graph(nodes and edges) and updating the result as the variable changes. Also we provide functionalities including defining the color of nodes and edges to make algorithms like BFS or DFS have a better expression. For example, the visited nodes could be red and and the unvisited ones could be blue.
Although we don't mean to implement a comprehensive library containing many graph algorithm, by using the primitive functionalities the programmers should be able to build up their own algorithms. But we will write a basic graph algorithm as demo to prove the quality of our language.

1.1 Imperative

Unlike Ocaml which is the programming language we will be using to implement the compiler, our language is imperative such that programmer can use statement to change the state of program. For example, if we want to let x be the smaller variable of x and y, we should write code like:

```
int x = 1;
int y = 2;
if (x > y) {
```

```
    int temp = x;
    x = y;
    y = temp;
}
```

1.2 Strong Typing

Our programming language is a strong typing language which means programmer must declare type of variables and it cannot be changed latter. Also, we don't allow implicit type conversion.

```
int x = 1;
// Compile-time error. It don't support implicit cast.
x = x + 1.0;
// The compiler still complains,
// even though there is no severe data loss.
float y = x;
```

2 Language Details

2.1 Data Types

Type	Description
int	4 bytes, integer
float	4 bytes, floating point number
char	1 bytes, character
bool	1 bytes, true/false value
array	Collection of elements of the same type
struct	Combination of multiple data members of different types
node	Struct containing position, size and color information of node
edge	Struct containing vertices, thickness and color information of edge
graph	Directed graph containing nodes and edges

2.2 Operators

Type	Description
Arithmetic	+,-,*,/,%
Assignment	=,+=,-=,*=,%=
Relational	==,!=,>,<,≥,≤
Logical	&&,‖,!

2.3 Control Flow, Functions and Comments

The control flow, functions and comments all follow c-style.

Control Flow

```
if (...) {...}
else if (...) {...}
else {...}

for (...) {...}

while (...) {...}
```

Functions

```
return_type function_name (arg_type1 arg1, arg_type2 arg2, ...) {...}
```

Comments

```
// single-line comment
/*
multi-line
comments
*/
```

2.4 Graph Bulit-In Functions

add_node(graph, node_id, node)	Add a node to the graph. node_id is used to identify this node in the graph. A node {x, y, radius, r, g, b} contains position information (x, y), size information radius, and color information (r, g, b).
add_edge(graph, edge)	Add an edge to the graph. An edge {n1, n2, thickness, r, g, b} contains vertices information (n1, n2), line thickness information thickness, color information (r, g, b). Edges are stored in 'adj' member of graph.
remove_node(graph, node_id)	Remove a node from graph.
remove_edge(graph, n1, n2)	Remove an edge linking node n1 and node n2 from graph.
change_node_rgb(graph, node_id, {r, g, b})	Change the color of a node in graph.
change_node_x(graph, node_id, x)	Change the x position of a node in graph.
change_node_y(graph, node_id, y)	Change the y position of a node in graph
change_edge_rgb(graph, n1, n2, r, g, b)	Change the color of an edge in graph.
show(graph1, graph2, ...)	Display graphs on a figure using openGL.

3 Example

```
void dfs(graph g, int v, bool visited[]) {
    visited[v] = true;
    change_node_rgb(g, v, {0, 100, 255});
    show(g);
    for (int i = 0; i < length(g.adj[v]); ++i) {
        if (!visited[g.adj[v][i]]) {
            dfs(g, g.adj[v][i], visited);
        }
    }
}
int main() {
    bool visited[4];
    graph g;
    for (int i = 0; i < 4; ++i) {
        add_node(g, i, {(i/2) * 4, (i%2) * 4, 1, 0, 0, 0});
    }
    add_edge(g, {0, 1, 0.2, 0, 0, 0});
    add_edge(g, {1, 3, 0.2, 0, 0, 0});
    add_edge(g, {0, 2, 0.2, 0, 0, 0});
    dfs(g, 0, visited);
}
```

The above program shows how to implement depth-first search algorithm and visualize its process. It will generate four figures one by one, with visited nodes marked as blue while the unvisited are marked as black.